Memory Leak (PART III)

AGENDA

I. Type of memory leak

II. Detect memory leak solution

III. Using static check tools

IV. Using runtime check tools

V. Analyze heap memory to detect memory leak

I - Type of memory leak

- **Syntactic Leak**

  Syntactic leak, this type of leak is very common and often seen in C/C++ or languages where programmers have to deal with memory allocation and release themselves. The main reason is when the developer allocates memory to use for some purpose, and then "forgets" to free the memory area, causing that memory area to be occupied and wasted. Specifically for C++ is new without a delete.

- **Semantic Leak**

  Semantic leak, this type of leak is present in all types of languages, even languages that are said to have a garbage collection mechanism. The cause of this leak is that the programmer uses containers and buffers to store temporary data, but after using it, forgets to release the memory.

II - Detect memory leak solution

Solutions	Pros	Cons
Static check tools • Cppcheck • Splint	✓ Simple, Fast, Easy to use	○ False positives ○ Cannot detect special cases
Memory debuggers • Valgrind	✓ Run-time check ✓ Precise, adequate results	○ Bad performance ○ Big memory overhead ○ Cannot run in QNX
Heap analysis • AddressSanitizer	✓ Can execute in almost ALl OS	○ Result depends on developer

III - Using Static check tools

Using Cppcheck to detect memory leak

Cppcheck is a static analysis tool that tries to completely avoid false warnings. A false warning is when the tool reports that there is an error even though there is no error.

```
hai@VN-MF10-NC100IN:/mnt/d/14.CBL/Memory leak/DemoPart3$ cppcheck leak1.c
Checking leak1.c ...
leak1.c:17:3: error: Array 'c[2]' accessed at index 2, which is out of bounds. [arrayIndexOutOfBounds]
 c[2] = 1;//Array access out of bounds
  ^
leak1.c:26:2: error: Memory leak: p [memleak]
 return 0;
 ^
```

III - Using Static check tools

Using Splint to detect memory leak

Splint can detect many memory management errors at compile time including using storage that may have been deallocated.

```
hai@VN-MF10-NC100IN:/mnt/d/14.CBL/Memory leak/DemoPart3$ splint leak1.c
Splint 3.1.2 --- 20 Feb 2018

leak1.c: (in function main)
leak1.c:13:2: Assignment of int to double: a = b
  To allow all numeric types to match, use +relaxtypes.
leak1.c:21:5: Test expression for if is assignment expression: a = 5
  The condition test is an assignment expression. Probably, you mean to use ==
  instead of =. If an assignment is intended, add an extra parentheses nesting
  (e.g., if ((a = b)) ...) to suppress this message. (Use -predassign to
  inhibit warning)
leak1.c:21:5: Test expression for if not boolean, type double: a = 5
  Test expression type is not boolean. (Use -predboolothers to inhibit warning)
leak1.c:26:11: Fresh storage p not released before return
  A memory leak has been detected. Storage allocated locally is not released
  before the last reference to it is lost. (Use -mustfreefresh to inhibit
  warning)
   leak1.c:15:2: Fresh storage p created
leak1.c:4:14: Parameter argc not used
  A function parameter is not used in the body of the function. If the argument
  is needed for type compatibility or future plans, use /*@unused@*/ in the
  argument declaration. (Use -paramuse to inhibit warning)
leak1.c:4:32: Parameter argv not used

Finished checking --- 6 code warnings
```

IV - Using runtime check tools

Using Valgrind to detect memory leak

The Valgrind tool suite provides a number of debugging and profiling tools that help you make your programs faster and more correct. The most popular of these tools is called Memcheck. It can detect many memory-related errors that are common in C and C++ programs and that can lead to crashes and unpredictable behaviour.

Use this command line:

```
valgrind --leak-check=yes myprog arg1 arg2
```

Memcheck is the default tool. The --leak-check option turns on the detailed memory leak detector.
Your program will run much slower (eg. 20 to 30 times) than normal, and use a lot more memory. Memcheck will issue messages about memory errors and leaks that it detects.

IV - Using runtime check tools

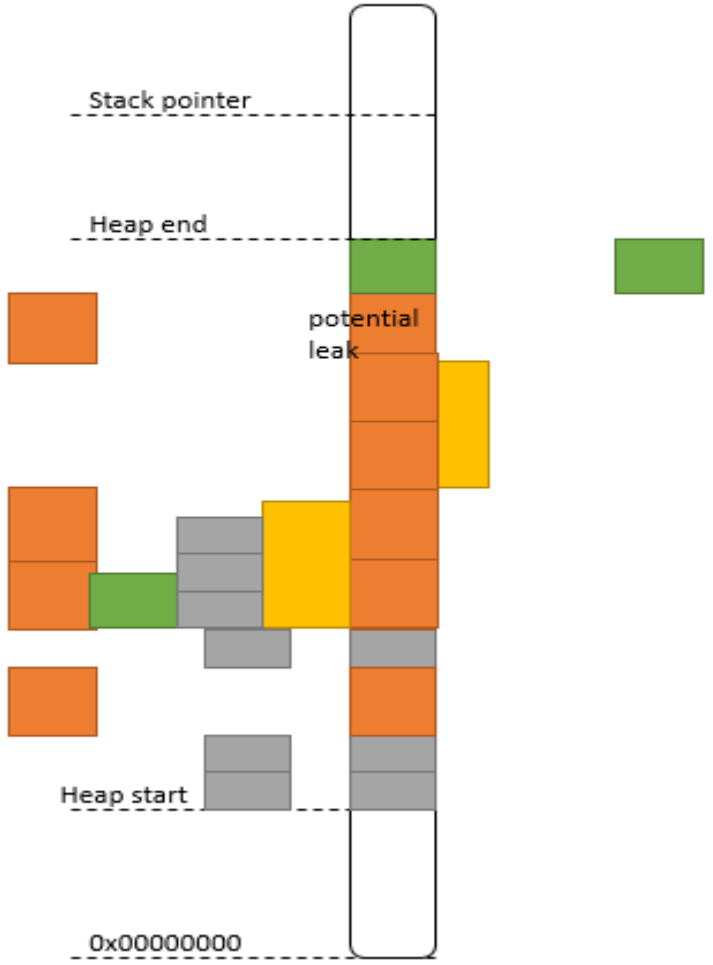
Using Valgrind to detect memory leak

```
hai@VN-MF10-NC100IN:/mnt/d/14.CBL/Memory leak/DemoPart3$ valgrind --leak-check=yes ./leak
==1383== Memcheck, a memory error detector
==1383== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1383== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==1383== Command: ./leak
==1383==
==1383== error calling PR_SET_PTRACER, vgdb might block
==1383== Invalid write of size 4
==1383==    at 0x10916B: f() (in /mnt/d/14.CBL/Memory leak/DemoPart3/leak)
==1383==    by 0x109180: main (in /mnt/d/14.CBL/Memory leak/DemoPart3/leak)
==1383==  Address 0x4a4d068 is 0 bytes after a block of size 40 alloc'd
==1383==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1383==    by 0x10915E: f() (in /mnt/d/14.CBL/Memory leak/DemoPart3/leak)
==1383==    by 0x109180: main (in /mnt/d/14.CBL/Memory leak/DemoPart3/leak)
==1383==
==1383==
==1383== HEAP SUMMARY:
==1383==     in use at exit: 40 bytes in 1 blocks
==1383==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==1383==
==1383== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1383==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==1383==    by 0x10915E: f() (in /mnt/d/14.CBL/Memory leak/DemoPart3/leak)
==1383==    by 0x109180: main (in /mnt/d/14.CBL/Memory leak/DemoPart3/leak)
==1383==
==1383== LEAK SUMMARY:
==1383==    definitely lost: 40 bytes in 1 blocks
==1383==    indirectly lost: 0 bytes in 0 blocks
==1383==      possibly lost: 0 bytes in 0 blocks
==1383==    still reachable: 0 bytes in 0 blocks
==1383==         suppressed: 0 bytes in 0 blocks
==1383==
==1383== For lists of detected and suppressed errors, rerun with: -s
==1383== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

V - Analyze heap memory to detect memory leak

Conceptual

- Find heaps of the process
- Collect all memory chunks of the heaps
- Classify the chunks based on their size
- Determine the potential leaks based on their count and their total size
- Investigate the chunk structure to find where in the source code it is create.



V - Analyze heap memory to detect memory leak

Motivation

- Dynamically allocations are allocated in heap
- Memory leaks are duplicated blocks
- The most duplicated blocks in heap are most likely to be leaks.

THANK YOU